Reasonable Highly Expressive Query Languages

Pierre Bourhis, Markus Krötzsch, Sebastian Rudolph

Reasonable Highly Expressive Query Languages

Pierre Bourhis[†] and **Markus Krötzsch**[‡] and **Sebastian Rudolph**[‡]
[†]CNRS CRIStAL UMR 9189 [‡]Technische Universität Dresden, Germany
pierre.bourhis@inria.fr {markus.kroetzsch,sebastian.rudolph}@tu-dresden.de

Abstract

Expressive query languages are gaining relevance in knowledge representation (KR), and new reasoning problems come to the fore. Especially query containment is interesting in this context. The problem is known to be decidable for many expressive query languages, but exact complexities are often missing. We introduce a new query language, *guarded queries* (GQ), which generalizes most known languages where query containment is decidable. GQs can be nested (more expressive), or restricted to linear recursion (less expressive). Our comprehensive analysis of the computational properties and expressiveness of (linear/nested) GQs also yields insights on many previous languages.

1 Introduction

The significance of query languages in KR is twofold. On the one hand, evaluating queries in the presence of a background ontology allows us to express more complex information needs, leading to the notion of *ontology-based query answering*. This topic has been studied for a wide range of ontology languages and many different query languages, including conjunctive queries [Calvanese *et al.*, 2007b; Eiter *et al.*, 2009] and (many variants of) regular path queries [Calvanese *et al.*, 2007a; 2009; Bienvenu *et al.*, 2014]. On the other hand, recursive queries can be used to "implement" reasoning, such that the query plays the role of a logical calculus that computes subsumptions [Xiao *et al.*, 2010; Krötzsch, 2011; Bischoff *et al.*, 2014].

In both application areas, we can see a tendency towards more and more powerful recursive queries. Recent works introduced several highly expressive query languages related to applications in KR: *Monadically Defined Queries* (MQs) [Rudolph and Krötzsch, 2013] and *Monadic Disjunctive SNP* queries (coMMSNP) [Bienvenu *et al.*, 2013]. Both can be viewed as fragments of (disjunctive) Datalog.

The proliferation of query languages and their uses in KR raises new questions. The complexity of ontology-based query answering has been studied from its inception, whereas the equally important question of relative expressiveness was studied only recently [Bienvenu *et al.*, 2013]. Another important question is the problem of *query containment*, where we consider two queries Q_1 and Q_2, and ask if every answer to Q_1 is also an answer to Q_2 over all possible inputs. Deciding query containment is relevant for query rewriting algorithms, where it needs to be checked if new queries are contained in previous ones to ensure termination. Further relevant applications are query optimization (finding a simpler yet equivalent query), and verification (checking that a query contains specific test cases). In addition, query containment has a range of applications in databases, e.g., in information integration and database integrity checking.

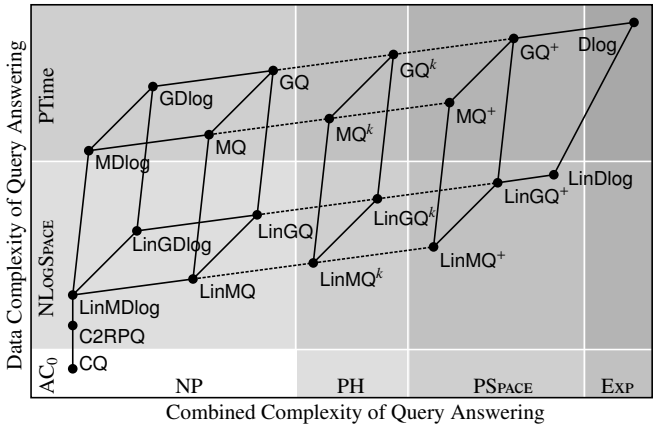

Figure 1: Query languages and complexities; languages higher up in the graph are more expressive

Although Datalog provides a useful framework for studying many recursive query languages, it does, unfortunately, not have a decidable query containment problem [Shmueli, 1987]. In contrast, the containment is known to be decidable for regular path queries, MQs, and coMMSNP queries. In the latter two cases, however, no upper complexity bound is known. Nevertheless, numerous results exist for various smaller query languages. For the following overview, recall that a predicate in a Datalog program is *intensional (IDB)* if it occurs in some rule head, and *extentional (EDB)* otherwise.

Non-recursive Datalog and unions of conjunctive queries A non-recursive Datalog program is equivalent to a (possibly exponential) union of conjunctive queries (UCQ), and thus expressible in first-order logic. Containment of Dat-

alog (Dlog) in UCQ is 2ExpTime-complete, while containment of Dlog by non-recursive Datalog is 3ExpTime-complete [Chaudhuri and Vardi, 1997]. Some restrictions for decreasing these complexities have been considered. Deciding if a *linear* Datalog program (LinDlog, where rule bodies contain at most one recursive predicate) is contained in a UCQ is ExpSpace-complete; complexity further decreases to PSpace when the linear Datalog program is monadic (LinMDlog, see below) [Chaudhuri and Vardi, 1994; 1997].

Monadic Datalog A monadic Datalog (MDlog) program is one with only unary IDB predicates. Containment between two MDlog programs is 2ExpTime-complete. The upper bound is well known since the 80's [Cosmadakis *et al.*, 1988], while the lower bound has been established only recently [Benedikt *et al.*, 2012]. Moreover, the containment of Dlog in MDlog is also decidable by a straightforward application of [Courcelle, 1991, Theorem 5.5].[1] So far, however, tight bounds have not been known for this case.

Guarded Datalog Guarded Datalog (GDlog) requires that, for each rule, the variables of the head should appear in a single EDB atom in the rule body. Such (frontier-)guarded rules have been known for a while [Calì *et al.*, 2008; Baget *et al.*, 2011], but their first use as a query language is recent [Bárány *et al.*, 2012]. GDlog is a proper extension of MDlog, since monadic rules can be rewritten into guarded rules [Bárány *et al.*, 2012]. Query containment for GDlog is 2ExpTime-complete, as it corresponds to a satisfiability problem for guarded negation fixed point logic [Bárány *et al.*, 2011].

Navigational Queries Conjunctive two-way regular path queries (C2RPQs) generalize conjunctive queries (CQs) by regular expressions over binary predicates [Florescu *et al.*, 1998; Calvanese *et al.*, 2003]. Variants of this idea are used in the RDF query language SPARQL 1.1 and the XML query language XPath. Roughly, C2RPQ is a conjunction of atoms of the form xLy where L is a two-way regular expression. A pair of nodes $\langle n_1, n_2\rangle$ is a valuation of the pair $\langle x, y\rangle$ if and only if there exists a path between n_1 and n_2 matching L. Containment of such queries is ExpSpace-complete [Florescu *et al.*, 1998; Calvanese *et al.*, 2003; Abiteboul and Vianu, 1999; Deutsch and Tannen, 2002], while containment of Dlog in C2RPQ is 2ExpTime-complete [Calvanese *et al.*, 2005].

Fragments of Monadic Second-Order Logic More recently, Monadically Defined Queries (MQs) and their nested version (MQ$^+$s) have been introduced as a proper generalization of MDlog that also captures (unions of) C2RPQs [Rudolph and Krötzsch, 2013]. MQs are expressible in both Dlog and monadic second-order logic, but (in contrast to these languages) feature a decidable query containment problem. The most general recent query language for which containment is known to be decidable is coMMSNP [Bienvenu *et al.*, 2013], a fragment of monadic second-order logic motivated by descriptive complexity. As opposed to the above languages, coMMSNP is a non-deterministic query language, closely related to disjunctive Datalog. A simple inspection of the definitions shows that the deterministic (disjunction-free) fragment, i.e., "Horn-coMMSNP", agrees with MQ.

[1] We thank Michael Benedikt for this observation.

In this paper, we further extend the known recursive query languages and at the same time settle all major questions related to the complexity of their query containment problems. Figure 1 gives an overview of all languages we consider, together with their respective query-answering complexities.

The main new query language we consider is called *guarded queries* (GQ), and is based on the use of frontier-guarded Datalog rules. GQ can be viewed as an extension of MQ, and is indeed inspired by a similar extension for coMMSNP [Bienvenu *et al.*, 2013]. GQ thus also generalizes frontier-guarded Datalog. We further introduce the nested and linear variant of GQ, and establish complexity results for query answering in all cases.

We then turn towards query containment. We obtain tight complexity bounds for (nested) GQs and many other query languages, which are summarized in Table 1. To show the upper bounds, we extend known automata-based approaches by a number of new techniques. Lower bounds are obtained by simulating space-bounded alternating Turing machines in a way that allows for an exponential increase in space with each nesting level. Finally, we also sketch how our results transfer to the case of linear Datalog, where many complexities can be slightly reduced.

In summary, our results settle open problems for (nested) MQs, painting a comprehensive and detailed picture of the state of the art in Datalog query containment. Full proofs can be found in an accompanying report [Bourhis *et al.*, 2015].

2 Preliminaries

We consider a standard language of first-order predicate logic, based on an infinite set $\mathbf{C}$ of *constant symbols*, an infinite set $\mathbf{P}$ of *predicate symbols*, and an infinite set $\mathbf{V}$ of first-order *variables*. Each predicate $p \in \mathbf{P}$ is associated with a natural number $\mathsf{ar}(p)$ called the *arity* of p. The list of predicates and constants forms the language's *signature* $\mathscr{S} = \langle \mathbf{P}, \mathbf{C}\rangle$. We generally assume $\mathscr{S} = \langle \mathbf{P}, \mathbf{C}\rangle$ to be fixed, and only refer to it explicitly if needed.

Formulae, Rules, and Queries A *term* is a variable $x \in \mathbf{V}$ or a constant $c \in \mathbf{C}$. We use symbols s, t to denote terms, x, y, z, v, w to denote variables, a, b, c to denote constants. Expressions like $\boldsymbol{t}, \boldsymbol{x}, \boldsymbol{c}$ denote finite lists of such entities. We use the standard predicate logic definitions of *atom* and *formula*, using symbols φ, ψ for the latter.

Datalog queries are defined over an extended signature with additional predicate symbols, called *IDB predicates*; all other predicates are called *EDB predicates*. A *Datalog rule* is a formula of the form $\forall \boldsymbol{x}, \boldsymbol{y}.\varphi[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \psi[\boldsymbol{x}]$ where φ and ψ are conjunctions of atoms, called the *body* and *head* of the rule, respectively, and where ψ only contains IDB predicates. We usually omit universal quantifiers when writing rules. Sets of Datalog rules are denoted $\mathbb{P}, \mathbb{R}, \mathbb{S}$. A set of Datalog rules is:

- *monadic* if all IDB predicates are of arity one;
- *frontier-guarded* if the body of every rule contains an atom $p(\boldsymbol{t})$ such that p is an EDB predicate and $\boldsymbol{t}$ contains all variables that occur in the rule's head;
- *linear* if each rule body has at most one IDB predicate.

A *conjunctive query* (CQ) is a formula $Q[\boldsymbol{x}] = \exists \boldsymbol{y}.\psi[\boldsymbol{x}, \boldsymbol{y}]$ where $\psi[\boldsymbol{x}, \boldsymbol{y}]$ is a conjunction of atoms; a *union of conjunctive queries* (UCQ) is a disjunction of such formulae. A *Datalog query* $\langle \mathbb{P}, Q \rangle$ consists of a set of Datalog rules $\mathbb{P}$ and a conjunctive query Q over IDB or EDB predicates (Q could be expressed as a rule in Datalog, but not in all restrictions of Datalog we consider). We write Dlog for the language of Datalog queries. A monadic Datalog query is one where $\mathbb{P}$ is monadic, and similarly for other restrictions. We use the query languages MDlog (monadic), GDlog (frontier-guarded), LinDlog (linear), and LinMDlog (linear, monadic).

Databases and Semantics We use the standard semantics of first-order logic (FOL). A *database instance* $\mathcal{I}$ *over a signature* $\mathscr{S} = \langle \mathbf{P}, \mathbf{C} \rangle$ consists of a set $\Delta^{\mathcal{I}}$ called *domain* and a function $\cdot^{\mathcal{I}}$ that maps constants $c \in \mathbf{C}$ to domain elements $c^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ and predicate symbols $p \in \mathbf{P}$ to relations $p^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^{\mathsf{ar}(p)}$, where $p^{\mathcal{I}}$ is the *extension* of p.

Given a database instance $\mathcal{I}$ and a formula $\varphi[\boldsymbol{x}]$ with free variables $\boldsymbol{x} = \langle x_1, \ldots, x_m \rangle$, the *extension* of $\varphi[\boldsymbol{x}]$ is the subset of $(\Delta^{\mathcal{I}})^m$ containing all those tuples $\langle \delta_1, \ldots, \delta_m \rangle$ for which $\mathcal{I}, \{x_i \mapsto \delta_i \mid 1 \leq i \leq m\} \models \varphi[\boldsymbol{x}]$. We denote this by $\langle \delta_1, \ldots, \delta_m \rangle \in \varphi^{\mathcal{I}}$ or by $\mathcal{I} \models \varphi(\delta_1, \ldots, \delta_m)$; a similar notation is used for all other types of query languages. Two formulae $\varphi[\boldsymbol{x}]$ and $\psi[\boldsymbol{x}]$ are called *equivalent* if their extensions coincide for every database instance $\mathcal{I}$.

The set of answers of a UCQ $Q[\boldsymbol{x}]$ over $\mathcal{I}$ is its extension. A Datalog program $\mathbb{P}$ is *satisfied* by database instance $\mathcal{I}'$ over the extended signature of EDB and IDB predicates, if all rules of $\mathbb{P}$ are satisfied by $\mathcal{I}'$ in the usual sense. The set of answers of a Datalog query $\langle \mathbb{P}, Q \rangle$ over $\mathcal{I}$ is the intersection of the extensions of Q over all extended database instances $\mathcal{I}'$ that satisfy $\mathbb{P}$ and agree with $\mathcal{I}$ on constants and EDB predicates. Datalog can also be defined as the least fixpoint of the inflationary evaluation of Q on $\mathcal{I}$ [Abiteboul *et al.*, 1994].

We do not require database instances to have a finite domain, since all of our results are valid in either case. This is due to the fact that every entailment of a Datalog program has a finite witness, and that all of our query languages are positive, i.e., that their answers are preserved under homomorphisms of database instances.

An important reasoning task on queries is to determine if a query contains another. In particular, a Datalog query $\langle \mathbb{P}, Q \rangle$ *is contained in* a Datalog query $\langle \mathbb{P}', Q' \rangle$, denoted $\langle \mathbb{P}, Q \rangle \sqsubseteq \langle \mathbb{P}', Q' \rangle$, iff for each database instance $\mathcal{I}$ over the signature of EDB predicates and constants, the set of answers of $\langle \mathbb{P}, Q \rangle$ over $\mathcal{I}$ is included in the set of answers of $\langle \mathbb{P}', Q' \rangle$ over $\mathcal{I}$.

3 Guarded Queries

Rudolph and Krötzsch [2013] introduced *monadically defined queries* (MQs[2]) as a generalization of conjunctive two-way regular path queries (C2RPQs) and monadic Datalog (MDlog) for which query containment is still decidable. The idea underlying this approach is that candidate query answers are checked by evaluating a monadic Datalog program, i.e., in contrast to the usual evaluation of Datalog queries, we start with a "guessed" answer that is the input to a Datalog program. To implement this, the candidate answer is represented by special constants λ that the Datalog program can refer to. This mechanism was called *flag & check*, since the special constants act as flags to indicate the answer to be checked.

Example 1. *A query that computes the transitive closure over a relation p can be defined as follows.*

$$p(\lambda_1, y) \rightarrow \mathtt{U}(y)$$
$$\mathtt{U}(y) \wedge p(y, z) \rightarrow \mathtt{U}(z)$$
$$\mathtt{U}(\lambda_2) \rightarrow \mathsf{hit}$$

One defines the answer of the query to contain all pairs $\langle \delta_1, \delta_2 \rangle$ *for which the rules entail* hit *when interpreting* λ_1 *as* δ_1 *and* λ_2 *as* δ_2.

The original approach used monadic Datalog for its close relationship to monadic second-order logic, which was the basis for showing decidability of query containment. In this work, however, we develop new techniques for showing the decidability (and exact complexity) of this problem directly. It is therefore suggestive to consider other types of Datalog programs for the "check" part. The next definition introduces the general approach for arbitrary Datalog programs, and defines interesting fragments by imposing further restrictions.

Definition 1. *Consider a signature* $\mathscr{S}$. *An FCP ("flag & check program") of arity m is a set of Datalog rules* $\mathbb{P}$ *with* $k \geq 0$ *IDB predicates* $\mathtt{U}_1, \ldots, \mathtt{U}_k$ *that may use the additional constant symbols* $\lambda_1, \ldots, \lambda_m \notin \mathscr{S}$ *and an additional nullary predicate symbol* hit. *An FCQ ("flag & check query") P is of the form* $\exists \boldsymbol{y}.\mathbb{P}(\boldsymbol{z})$, *where* $\mathbb{P}$ *is an FCP of arity* $|\boldsymbol{z}|$ *and all variables in* $\boldsymbol{y}$ *occur in* $\boldsymbol{z}$. *The variables* $\boldsymbol{x}$ *that occur in* $\boldsymbol{z}$ *but not in* $\boldsymbol{y}$ *are the* free variables *of P.*

Let $\mathcal{I}$ *be a database instance over* $\mathscr{S}$. *The* extension $\mathbb{P}^{\mathcal{I}}$ *of* $\mathbb{P}$ *is the set of all tuples* $\langle \delta_1, \ldots, \delta_m \rangle \in (\Delta^{\mathcal{I}})^m$ *such that every database instance* $\mathcal{I}'$ *that extends* $\mathcal{I}$ *to the signature of* $\mathbb{P}$ *and that satisfies* $\langle \lambda_1^{\mathcal{I}'}, \ldots, \lambda_m^{\mathcal{I}'} \rangle = \langle \delta_1, \ldots, \delta_m \rangle$ *also entails* hit. *The semantics of FCQs is defined in the obvious way based on the extension of FCPs.*

A GQ *is an FCQ* $\exists \boldsymbol{y}.\mathbb{P}(\boldsymbol{z})$ *such that* $\mathbb{P}$ *is frontier-guarded. Similarly, we define* MQ *(monadic),* LinMQ *(linear, monadic), and* LinGQ *(linear, frontier-guarded) queries.*

In contrast to Rudolph and Krötzsch [2013], we do not define monadic queries as conjunctive queries of FCPs, but we merely allow existential quantification to project some of the FCP variables. Proposition 1 below shows that this does not reduce expressiveness.

We generally consider monadic Datalog as a special case of frontier-guarded Datalog. Monadic Datalog rules do not have to be frontier-guarded. A direct way to obtain a suitable guard is to assume that there is a unary domain predicate that contains all (relevant) elements of the domain of the database instance. However, it already suffices to require *safety* of Datalog rules, i.e., that the variable in the head of a rule must also occur in the body. Then every element that is inferred to belong to an IDB relation must also occur in some EDB relation. We can therefore add single EDB guard atoms to each rule in all possible ways without modifying the semantics. This is a polynomial operation, since all variables in the guards are fresh, other than the single head variable that we

[2]Here we shorten the original acronym MODEQ to MQ.

want to guard. We therefore find, in particular, that GQ captures the expressiveness of MQ. The converse is not true, as the following example illustrates.

Example 2. *The following* 4*-ary* LinGQ *generalizes Example 1 by checking for the existence of two parallel p-chains of arbitrary length, where each pair of elements along the chains is connected by a relation q, like the steps of a ladder.*

$$q(\lambda_1, \lambda_2) \rightarrow \mathtt{U}_q(\lambda_1, \lambda_2)$$
$$\mathtt{U}_q(x, y) \wedge p(x, x') \wedge p(y, y'), q(x', y') \rightarrow \mathtt{U}_q(x', y')$$
$$\mathtt{U}_q(\lambda_3, \lambda_4) \rightarrow \mathsf{hit}$$

One might assume that the following MQ *is equivalent:*

$$q(\lambda_1, \lambda_2) \rightarrow \mathtt{U}_1(\lambda_1)$$
$$q(\lambda_1, \lambda_2) \rightarrow \mathtt{U}_2(\lambda_2)$$
$$\mathtt{U}_1(x) \wedge \mathtt{U}_2(y) \wedge p(x, x') \wedge p(y, y'), q(x', y') \rightarrow \mathtt{U}_1(x')$$
$$\mathtt{U}_1(x) \wedge \mathtt{U}_2(y) \wedge p(x, x') \wedge p(y, y'), q(x', y') \rightarrow \mathtt{U}_2(y')$$
$$\mathtt{U}_1(\lambda_3) \wedge \mathtt{U}_2(\lambda_4) \rightarrow \mathsf{hit}$$

However, the latter query also matches structures that are not ladders. For example, the following database yields the answer $\langle a, b, c, d\rangle$*, although there is no corresponding ladder structure:* $\{q(a, b), p(a, c), p(b, e), q(c, e), p(a, e'), p(b, d), q(e', d)\}$*. One can extend the* MQ *to avoid this case, but any such fix is "local" in the sense that a sufficiently large ladder-like structure can trick the query.*

Rudolph and Krötzsch [2013] showed that monadically defined queries can be expressed both in Datalog and in monadic second-order logic. While we lose the connection to monadic second-order logic with GQs, the expressibility in Datalog remains. The encoding is based on the intuition that the choice of the candidate answers for $\boldsymbol{\lambda}$ "contextualizes" the inferences of the Datalog program. To express this without special constants, we can store this context information in predicates of suitably increased arity.

Example 3. *The* 4*-ary* LinGQ *of Example 2 can be expressed with the following Datalog query. For brevity, let* $\boldsymbol{y}$ *be the variable list* $\langle y_1, y_2, y_3, y_4\rangle$*, which provides the context for the IDB facts we derive.*

$$q(y_1, y_2) \rightarrow \mathtt{U}^+_q(y_1, y_2, \boldsymbol{y})$$
$$\mathtt{U}_q(x, y, \boldsymbol{y}) \wedge p(x, x') \wedge p(y, y'), q(x', y') \rightarrow \mathtt{U}^+_q(x', y', \boldsymbol{y})$$
$$\mathtt{U}_q(y_3, y_4, \boldsymbol{y}) \rightarrow \mathsf{goal}(\boldsymbol{y})$$

This result is obtained by a straightforward extension of the translation algorithm for MQs *[Rudolph and Krötzsch, 2013], which may not produce the most concise representation. Also note that the first rule in this program is not safe, since* y_3 *and* y_4 *occur in the head but not in the body. According to the semantics we defined, such variables can be bound to any element in the active domain of the given database instance (i.e., they behave as if bound by a unary* domain *predicate).*

This observation justifies that we consider MQs, GQs, etc. as Datalog fragments. It is worth noting that the translation does not change the number of IDB predicates in the body of rules, and thus preserves linearity. The relation to (linear) Datalog also yields some complexity results for query answering; we will discuss these at the end of the next section, after introducing nested variants our query languages.

4 Nested Queries

Every query language gives rise to a nested language, where we allow the use of nested queries as if they were predicates. Sometimes, this does not lead to a new query language (like for CQ and Dlog), but often it affects complexities and/or expressiveness. It has been shown that both are increased when moving from MQs to their nested variants [Rudolph and Krötzsch, 2013]. We will see that nesting also has strong effects on the complexity of query containment.

Definition 2. *We define k-nested FCPs inductively. A* 1*-nested FCP is an FCP. A* $k+1$*-nested FCP is an FCP that may use k-nested FCPs of arity m instead of predicate symbols of arity m in rule bodies. The semantics of nested FCPs is immediate based on the extension of FCPs. A k-nested FCQ P is of the form* $\exists \boldsymbol{y}.\mathbb{P}(\boldsymbol{z})$*, where* $\mathbb{P}$ *is a k-nested FCP of arity* $|\boldsymbol{z}|$ *and all variables in* $\boldsymbol{y}$ *occur in* $\boldsymbol{z}$*.*

A k-nested GQ *query is a k-nested frontier-guarded FCQ. For the definition of* frontier-guarded*, we still require EDB predicates in guards: subqueries cannot be guards. The language of k-nested* GQ *queries is denoted* GQ^k*; the language of arbitrarily nested* GQ *queries is denoted* GQ^+*.*

Similarly, we define languages MQ^k *and* MQ^+ *(monadic),* LinMQ^k *and* LinMQ^+ *(linear, monadic), and* LinGQ^k *and* LinGQ^+ *(linear, frontier-guarded).*

Note that nested queries can use the same additional symbols (predicates and constants); this does not lead to any semantic interactions, however, as the interpretation of the special symbols is "private" to each query. To simplify notation, we assume that distinct (sub)queries always contain distinct special symbols. The relationships of the query languages we introduced here are summarized in Figure 1, where upwards links denote increased expressiveness. An interesting observation that is represented in this figure is that linear Datalog is closed under nesting:

Theorem 3. $\mathsf{LinDlog} = \mathsf{LinDlog}^+$.

Another kind of nesting that does not add expressiveness is the nesting of FCQs in UCQs. Indeed, it turns out that (nested) FCQs can internalize arbitrary conjunctions and disjunctions of FCQs (of the same nesting level). This even holds when restricting to linear rules.

Proposition 1. *Let P be a positive query, i.e., a Boolean expression of disjunctions and conjunctions, of* LinMQ^k *queries with* $k \geq 1$*. Then there is a* LinMQ^k *query* P' *of size polynomial in P that is equivalent to P. Analogous results hold when replacing* LinMQ^k *by* MQ^k*,* GQ^k*, or* LinMQ^k *queries.*

Query answering for MQs has been shown to be NP-complete (combined complexity) and P-complete (data complexity). For MQ^+, the combined complexity increases to PSpace while the data complexity remains the same. These results can be extended to GQs. We also note the complexity for frontier-guarded Datalog, for which we are not aware of any published result.

Theorem 4. *The combined complexity of evaluating* GQ *queries over a database instance is* NP*-complete. The same holds for* GDlog *queries. The combined complexity of evaluating* GQ^+ *queries is* PSpace*-complete. The data complexity is* P*-complete for* GDlog*,* GQ*, and* GQ^+*.*

The lower bounds in the previous case follow from know results for MQs. Particularly, the hardness proof for nested MQs also shows that queries of a fixed nesting level can encode the validity problem for quantified boolean formulae with a certain number of quantifier alternations; this explains why we show the combined complexity of MQ^k to be in the Polynomial Hierarchy in Figure 1. A modification of this hardness proof of Rudolph and Krötzsch [2013] yields the same results for the combined complexities in the linear cases; matching upper bounds follow from Theorem 4.

Theorem 5. *The combined complexity of evaluating* LinMQ*,* LinGDlog*, or* LinGQ *queries over a database instance is* NP*-complete. The combined complexity of evaluating* LinMQ^+ *or* LinGQ^+ *queries is* PSpace*-complete. The data complexity is* NLogSpace*-complete for all of these query languages.*

5 Complexity of Query Containment

In this section, we first discuss an automata-based way to decide query containment, yielding upper complexity bounds.

We first recall a general technique of reducing query containment to the containment problem for (tree) automata [Chaudhuri and Vardi, 1997]. In spite of several extensions we need for λ-terms and nesting, our proofs still follow the same basic approach. An introduction to tree automata is included in the report [Bourhis *et al.*, 2015].

A common way to describe the answers of a Dlog query $P = \langle \mathbb{P}, p \rangle$ is to consider its *expansion trees*. Intuitively speaking, the goal atom $p(\boldsymbol{x})$ can be rewritten by applying rules of $\mathbb{P}$ in a backward-chaining manner until all IDB predicates have been eliminated, resulting in a CQ. The answers of P coincide with the (infinite) union of answers to the CQs obtained in this fashion. The rewriting itself gives rise to a tree structure, where each node is labeled by the instance of the rule that was used in the rewriting, and the leaves are instances of rules that contain only EDB predicates in their body. The set of all expansion trees provides a regular description of P that we exploit to decide containment.

To formalize this approach, we describe the set of all expansion trees as a tree language, i.e., as a set of trees with node labels from a finite alphabet. The number of possible labels of nodes in expansion trees is unbounded, since rules are instantiated using fresh variables. To obtain a finite alphabet of labels, one limits the number of variables and thus the overall number of possible rule instantiations [Chaudhuri and Vardi, 1997]. The set of *proof trees* obtained in this way is a regular tree language that can be described by an automaton $\mathcal{A}_P$. In order to use $\mathcal{A}_P$ to decide containment of P by another query P', we construct an automaton $\mathcal{A}_{P \sqsubseteq P'}$ that accepts all proof trees of P that are "matched" by P'. Indeed, every proof tree induces a *witness*, i.e., a minimal matching database instance, and one can check whether or not P' can produce the same query answer on this instance. If this is the case for all proof trees of P, then containment is shown.

Our first result provides the upper bound for deciding containment of GQ queries. In fact, the result extends to arbitrary Dlog queries on the left-hand side.

Theorem 6. *Containment of* Dlog *queries in* GQ *queries can be decided in* 3ExpTime.

The proof of this result requires a number of new techniques on top of the established methods. We are looking for an automaton $\mathcal{A}_{P \sqsubseteq P'}$ that accepts proof trees of P where the underlying witness is also accepted by P'. As a first step, we construct an automaton $\mathcal{A}_{P,\rho}$ that verifies that a single rule ρ of P' can be applied in a specific way to derive one specific conclusion. Since proof trees reuse variables to obtain a finite alphabet, the conclusion of the rule is an atom $p(\boldsymbol{v})$ referring to variables $\boldsymbol{v}$ that are ambiguous if we do not know exactly which place in the tree we are referring to. Therefore the input of $\mathcal{A}_{P,\rho}$ is a proof tree of P with two kinds of additional information added to the labels: (a) the interpretation of the λ constants that is used, and (b) the expected conclusion of the rule. $\mathcal{A}_{P,\rho}$ is a top-down tree automaton of exponential size.

We want to combine many automata of the form $\mathcal{A}_{P,\rho}$ to verify complete derivations of P' rather than single rule applications. In this case, we cannot add information about the expected conclusion $p(\boldsymbol{v})$ to the tree, since there are unboundedly many conclusions during one run. Instead, we encode the conclusion by considering auomtata $\mathcal{A}^+_{P,\rho,\boldsymbol{v}}$ that can start their run not just from the root, but from some node within the tree where all variables $\boldsymbol{v}$ occur with the same meaning as in the conclusion $p(\boldsymbol{v})$ (this is a single node due to guardedness). Starting in the middle of the tree makes it necessary to consider both nodes below and above the current position, and $\mathcal{A}^+_{P,\rho,\boldsymbol{v}}$ thus needs to be an *alternating 2-way tree automaton*.

An automaton $\mathcal{A}^+_{P \sqsubseteq P'}$ that verifies a complete derivation of P' on a proof tree of P is obtained by "concatenating" automata of the form $\mathcal{A}^+_{P,\rho,\boldsymbol{v}}$. $\mathcal{A}^+_{P \sqsubseteq P'}$ is an alternating 2-way automaton that is exponential in size. The trees accepted by $\mathcal{A}^+_{P \sqsubseteq P'}$ still need to contain information about the interpretation of λ-constants. Using a well-known construction, we obtain an exponentially larger (1-way) top-down tree automaton $\mathcal{A}'_{P \sqsubseteq P'}$ that accepts the same trees. This automaton of double exponential size can finally be changed into the automaton $\mathcal{A}_{P \sqsubseteq P'}$ that does not require λ-annotations—a polynomial transformation. We finish with a doubly-exponential automaton $\mathcal{A}_{P \sqsubseteq P'}$. Checking containment in $\mathcal{A}_P$ is an exponential process, leading to the claimed 3ExpTime result.

We can modify this proof to obtain another interesting result for the case of frontier-guarded Datalog. If P is a GDlog query, which does not use any special constants λ, we can directly construct a complement tree automaton $\bar{\mathcal{A}}_{P \sqsubseteq P'}$ that is only doubly exponential [Cosmadakis *et al.*, 1988, Theorem A.1]. Containment can then be checked by checking the non-emptiness of $\mathcal{A}_P \cap \bar{\mathcal{A}}_{P \sqsubseteq P'}$, which is possible in polynomial time, leading to a 2ExpTime algorithm.

Theorem 7. *Containment of* Dlog *queries in* GDlog *queries can be decided in* 2ExpTime.

This generalizes an earlier result of Cosmadakis *et al.* [1988] for monadic Datalog, using another, direct proof.

To lift our results to nested queries, we further extend the ideas developed in the non-nested case. Nested queries are similar to IDB predicates whose validity we need to check using automata. To do this, we first construct alternating two-way tree automata $\mathcal{A}^+_{P,Q,\theta}$ that verify a match of query Q on a tree that is annotated with the expected values of the λ-constants. To remove the need for this annotation when ver-

	UCQ,LinMDlog, MDlog, LinGDlog, GDlog	LinMQ^k, LinGQ^k	MQ^k, GQ^k	LinMQ$^+$,MQ$^+$, LinGQ$^+$,GQ$^+$	Dlog
LinMQ	PSpace-h [Chaudhuri and Vardi, 1994] ExpSpace [Bourhis *et al.*, 2015]	kExpSpace-h [Bourhis *et al.*, 2015] ($k+1$)ExpSpace [Bourhis *et al.*, 2015]	($k+1$)ExpSpace-c [Bourhis *et al.*, 2015]	Nonelementary [Bourhis *et al.*, 2015]	Undecidable [Abiteboul *et al.*, 1994]
LinGDlog, LinMQ^n ($n \geq 2$), LinMQ$^+$, LinGQ$^+$, LinGQ^n, LinDlog	ExpSpace-c [Bourhis *et al.*, 2015]	($k+1$)ExpSpace-c [Bourhis *et al.*, 2015]	($k+1$)ExpSpace-c [Bourhis *et al.*, 2015]	Nonelementary [Bourhis *et al.*, 2015]	Undecidable [Abiteboul *et al.*, 1994]
MDlog, GDlog, MQ^n, GQ^n, MQ$^+$, GQ$^+$, Dlog	2ExpTime-c [Benedikt *et al.*, 2012], [Chaudhuri and Vardi, 1997]\ [Cosmadakis *et al.*, 1988], [Th.7]	($k+2$)ExpTime-c [Th.9]\[Th.8]	($k+2$)ExpTime-c [Th.9]\[Th.8]	Nonelementary [Th.9]	Undecidable [Shmueli, 1987]

Table 1: Summary of the known complexities of query containment for several Datalog fragments; sources for each claim are shown in square brackets, using \ to separate sources for lower and upper complexity bounds, respectively

ifying subqueries as part of a longer run, we can again transform $\mathcal{A}^+_{P,Q,\theta}$ into a tree automaton (exponential), and project away the $\boldsymbol{\lambda}$-annotations (polynomial). The resulting automaton $\mathcal{A}_{P,Q}$ is analogous to the above tree automaton $\mathcal{A}_{P,\rho}$. The rest of the proof uses similar constructions as before. The exponential transformation from $\mathcal{A}^+_{P,Q,\theta}$ to $\mathcal{A}_{P,Q}$ is the reason for the exponential complexity increase in each nesting level.

Theorem 8. *Containment of* Dlog *queries in* GQ^k *queries can be decided in* ($k+2$)ExpTime.

To obtain matching lower bounds, we provide direct encodings of Alternating Turing Machines (ATMs) with a fixed space bound. In the context of query containment, this is done by defining a pair of queries P_1 and P_2 such that P_1 matches all structures that encode a sequence (or tree) of (unrelated) Turing machine configurations, while P_2 matches all such structures that do not correctly encode a run of the given TM (i.e., P_2 detects encoding errors). Then any structure that is matched by P_1 but not by P_2 encodes a terminating ATM run, such that the ATM halts iff P_1 is not contained in P_2.

To obtain hardness results for arbitrary towers of exponential functions, all of our constructions use existing queries to construct larger queries. For example, a query SameCell$[x, y]$ is defined to match the cells in neighboring configurations that are located at the same position of the ATM tape. This query becomes more and more complex (and more and more nested) as we go to exponentially larger tapes, but the construction of the queries needed for the next level always follows the same pattern. In spite of this efficient presentation, the complete ATM encoding requires significant space, and we must refer to the technical report for the details.

Theorem 9. *Deciding containment of* MDlog *queries in* MQ^k *queries is hard for* ($k+2$)ExpTime.

Note that the statement includes the 3ExpTime-hardness for containment of MQs as a special case.

A range of further results can be obtained by considering linear Datalog instead of Datalog in the role of the contained query. This tends to reduce complexity since one can focus on linear derivations, which can be described by word automata instead of tree automata. Accordingly, many ExpTime problems are reduced to PSpace, and all previous complexities for ($k+2$)ExpTime translate into results for ($k+1$)ExpSpace accordingly. Our ATM constructions are replaced by regular TM constructions, and we obtain tight bounds in most cases. The only exception is containment of LinMQ in LinMQ^k, where our lower bounds are one exponential below the upper bounds. The exact complexity remains open.

6 Conclusions

We have studied the most expressive fragments of Datalog for which query containment is known to be decidable, and we provided exact complexities for query answering and containment in most cases. Our results are summarized in Table 1. While containment tends to be nonelementary for nested queries, we have identified tight exponential complexity hierarchies depending on nesting depth. Our results settle several open problems for known query languages: the complexity of query containment for MQ and MQ$^+$, the complexity of query containment of Dlog in GDlog, and the expressivity of nested LinDlog.

Moreover, we have introduced new query languages based on frontier-guarded Datalog, showing that most complexities are unaffected by this extension.

A few small questions remain open. First, our results are not tight for linear MQs. This case is closely related to conjunctive regular path queries, and inspiration might be drawn from recent results in this field [Reutter, 2013]. Another question is about the role of constants, which we use heavily in some of our hardness proofs. For the case of (linear) monadic Datalog without constants, we conjecture that containment complexities are reduced by one exponential each.

Promising directions for future research include the study of practical containment algorithms, since our automata-based techniques do not lend themselves to implementation yet. Another interesting topic is the search for suitable queries that contain a given query. A special case of this is the *boundedness* problem, where we try to find a UCQ that contains a given Datalog program. This can be addressed by similar automata-based constructions [Cosmadakis *et al.*, 1988]. Besides boundedness, one can also ask more general questions of *rewritability*, e.g., whether some Datalog program can be expressed in monadic Datalog or in a regular path query.

Acknowledgements Pierre Bourhis was partially supported by the INRIA North European associate team *Integrating Linked Data*. Markus Krötzsch was supported by the DFG in Emmy Noether grant "DIAMOND" (KR 4381/1-1).

References

[Abiteboul and Vianu, 1999] Serge Abiteboul and Victor Vianu. Regular path queries with constraints. *J. Comput. Syst. Sci.*, 58(3):428–452, 1999.

[Abiteboul *et al.*, 1994] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley, 1994.

[Baget *et al.*, 2011] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9–10):1620–1654, 2011.

[Bárány *et al.*, 2011] Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP (2)*, volume 6756 of *LNCS*, pages 356–367. Springer, 2011.

[Bárány *et al.*, 2012] Vince Bárány, Balder ten Cate, and Martin Otto. Queries with guarded negation. *PVLDB*, 5(11):1328–1339, 2012.

[Benedikt *et al.*, 2012] Michael Benedikt, Pierre Bourhis, and Pierre Senellart. Monadic datalog containment. In *Proc. 39th Int. Coll. on Automata, Languages, and Programming (ICALP'12)*, pages 79–91, 2012.

[Bienvenu *et al.*, 2013] Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. In Richard Hull and Wenfei Fan, editors, *Proc. 32nd Symp. on Principles of Database Systems (PODS'13)*, pages 213–224. ACM, 2013.

[Bienvenu *et al.*, 2014] Meghyn Bienvenu, Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. Nested regular path queries in description logics. In Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter, editors, *Proc. 14th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'14)*. AAAI Press, 2014.

[Bischoff *et al.*, 2014] Stefan Bischoff, Markus Krötzsch, Axel Polleres, and Sebastian Rudolph. Schema-agnostic query rewriting for SPARQL 1.1. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig A. Knoblock, Denny Vrandečić, Paul T. Groth, Natasha F. Noy, Krzysztof Janowicz, and Carole A. Goble, editors, *Proc. 13th Int. Semantic Web Conf. (ISWC'14)*, volume 8796 of *LNCS*, pages 584–600. Springer, 2014.

[Bourhis *et al.*, 2015] Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. Reasonable highly expressive query languages: Extended technical report. Available at `https://ddll.inf.tu-dresden.de/web/Techreport3020`, 2015.

[Calì *et al.*, 2008] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In Gerhard Brewka and Jérôme Lang, editors, *Proc. 11th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 70–80. AAAI Press, 2008.

[Calvanese *et al.*, 2003] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.

[Calvanese *et al.*, 2005] Diego Calvanese, Giuseppe De Giacomo, and Moshe Y. Vardi. Decidable containment of recursive queries. *Theor. Comput. Sci.*, 336(1):33–56, 2005.

[Calvanese *et al.*, 2007a] Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Answering regular path queries in expressive description logics: An automata-theoretic approach. In *Proc. 22nd AAAI Conf. on Artificial Intelligence (AAAI'07)*, pages 391–396. AAAI Press, 2007.

[Calvanese *et al.*, 2007b] Diego Calvanese, Guiseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[Calvanese *et al.*, 2009] Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Regular path queries in expressive description logics with nominals. In Craig Boutilier, editor, *Proc. 21st Int. Joint Conf. on Artificial Intelligence (IJCAI'09)*, pages 714–720. IJCAI, 2009.

[Chaudhuri and Vardi, 1994] Surajit Chaudhuri and Moshe Y. Vardi. On the complexity of equivalence between recursive and nonrecursive Datalog programs. In *Proc. 13th Symp. on Principles of Database Systems (PODS'93)*, pages 107–116, 1994.

[Chaudhuri and Vardi, 1997] Surajit Chaudhuri and Moshe Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. *J. of Comput. Syst. Sci.*, 54(1):61–78, 1997.

[Cosmadakis *et al.*, 1988] Stavros Cosmadakis, Haim Gaifman, Paris Kanellakis, and Moshe Vardi. Decidable optimization problems for database logic programs. In *Proc. 20th Annual ACM Symp. on Theory of Computing (STOC'88)*, pages 477–490. ACM, 1988.

[Courcelle, 1991] Bruno Courcelle. Recursive queries and context-free graph grammars. *Theor. Comput. Sci.*, 78(1):217–244, 1991.

[Deutsch and Tannen, 2002] Alin Deutsch and Val Tannen. Optimization properties for classes of conjunctive regular path queries. In *Revised Papers from the 8th Int. Workshop on Database Programming Languages (DBPL'01)*, pages 21–39. Springer, 2002.

[Eiter *et al.*, 2009] Thomas Eiter, Carsten Lutz, Magdalena Ortiz, and Mantas Simkus. Query answering in description logics with transitive roles. In Craig Boutilier, editor, *Proc. 21st Int. Joint Conf. on Artificial Intelligence (IJCAI'09)*, pages 759–764. IJCAI, 2009.

[Florescu *et al.*, 1998] Daniela Florescu, Alon Levy, and Dan Suciu. Query containment for conjunctive queries with regular expressions. In Alberto O. Mendelzon and Jan Paredaens, editors, *Proc. 17th Symp. on Principles of Database Systems (PODS'98)*, pages 139–148. ACM, 1998.

[Krötzsch, 2011] Markus Krötzsch. Efficient rule-based inferencing for OWL EL. In Toby Walsh, editor, *Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11)*, pages 2668–2673. AAAI Press/IJCAI, 2011.

[Reutter, 2013] Juan L. Reutter. Containment of nested regular expressions. *CoRR*, abs/1304.2637, 2013.

[Rudolph and Krötzsch, 2013] Sebastian Rudolph and Markus Krötzsch. Flag & check: Data access with monadically defined queries. In Richard Hull and Wenfei Fan, editors, *Proc. 32nd Symp. on Principles of Database Systems (PODS'13)*, pages 151–162. ACM, 2013.

[Shmueli, 1987] O. Shmueli. Decidability and expressiveness aspects of logic queries. In *Proc. 6th Symp. on Principles of Database Systems (PODS'87)*, pages 237–249. ACM, 1987.

[Xiao *et al.*, 2010] Guohui Xiao, Stijn Heymans, and Thomas Eiter. DReW: a reasoner for Datalog-rewritable description logics and dl-programs. In Thomas Eiter, Adil El Ghali, Sergio Fernández, Stijn Heymans, Thomas Krennwallner, and François Lévy, editors, *Proc. 1st Int. Workshop on Business Models, Business Rules and Ontologies (BuRO'10)*, pages 1–14. ONTORULE Project, 2010.